VamPy - Bug #1868
Crash on cleanup when presented with failed plugin that redefines Feature

2019-01-11 01:31 PM - Chris Cannam

Status:	New	**Start date:**	2019-01-11
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			

Description

Reproduceable on macOS and Linux with VamPy 2.0 or repo tip, system Python 2.7.

- Ensure that you have vampy.so or vampy.dylib in a Vamp plugin directory
- Place the following in a file called Test.py in a Vamp plugin directory. Note that this defines its own Feature class (colliding with the VamPy one), and also takes too few args to __init__ of the Test class for it to be usable as a VamPy plugin:

```
class Feature(object):
    def __init__(self):
        print "I am a Feature"

class Test:
    def __init__(self):
        print "I am a Test"
```

- Run the Piper Vamp plugin server (as used by SV), e.g. with the following incantation

```
$ echo '{"method": "list"}' | piper-vamp-simple-server -d json
```

- Observe that the server crashes before printing the return value of the list method call.

If you either remove the Feature class from Test.py, or make the Test__init__ method take two arguments, then it doesn't crash. If Test.py is the only Python file present in the Vamp path, it also doesn't crash - there must be at least one valid Vamp plugin as well.

Stack trace on Linux via valgrind:

```
==18262== Process terminating with default action of signal 11 (SIGSEGV): dumping core
==18262==  Access not within mapped region at address 0x0
==18262==    at 0x76E6054: type_dealloc.lto_priv.57 (in /usr/lib/libpython2.7.so.1.0)
==18262==    by 0x7642898: PyDict_Clear (in /usr/lib/libpython2.7.so.1.0)
==18262==    by 0x487B231: PyExtensionManager::cleanModule() const (PyExtensionManager.cpp:218)
==18262==    by 0x487A80A: PyExtensionManager::~PyExtensionManager() (PyExtensionManager.cpp:91)
==18262==    by 0x517B996: __cxa_finalize (in /usr/lib/libc-2.28.so)
==18262==    by 0x484CC27: ??? (in /home/cannam/vamp/vampy.so)
==18262==    by 0x4015154: _dl_close_worker (in /usr/lib/ld-2.28.so)
==18262==    by 0x4015AA1: _dl_close (in /usr/lib/ld-2.28.so)
==18262==    by 0x5278F56: _dl_catch_exception (in /usr/lib/libc-2.28.so)
==18262==    by 0x5278FF2: _dl_catch_error (in /usr/lib/libc-2.28.so)
==18262==    by 0x4E0F8BE: ??? (in /usr/lib/libdl-2.28.so)
==18262==    by 0x4E0F287: dlclose (in /usr/lib/libdl-2.28.so)
```

The crash occurs in PyExtensionManager::cleanModule, but it seems to be a consequence of something that happens in the preceding call to PyExtensionManager::cleanLocalNamespace. If you modify cleanLocalNamespace so as to return immediately when the namespace name matches "Test", then the crash does not occur. Alternatively you can modify PyExtensionManager::m_exposedNames (the set of names cleaned from the local namespace) and remove Feature to avert this crash, though presumably it would still crash if the "plugin" redefined one of the other classes.

History

#1 - 2019-01-11 01:34 PM - Chris Cannam

- Description updated

#2 - 2019-01-11 01:52 PM - Chris Cannam

- Description updated

#3 - 2019-01-11 02:38 PM - Chris Cannam

The crash happens the third time cleanModule is called. Since cleanModule is only called from the PyExtensionManager destructor, and there is only one static PyExtensionManager, I guess it must be the third time the plugin dll has been unloaded.

The order of events in my test case appears to be (when asked to produce the list of plugin static data):

1. Call vampGetPluginDescriptor(2, n) for each n. The first call has isPythonInitialised is 0 and haveScannedPlugins is 0. Subsequent calls have both equal to 1. Note that the scan, which happens during the first call, fails to detect that the plugin won't load - it just looks for plausible filename / classname combinations - so updateLocalNamespace is called for the bogus plugin just as for all the rest at this point. I have five real VamPy plugins installed plus this bogus one, so I see n from 0 to 5. (The call with n equal to 5 yields a null return to show that the list has ended; one of the earlier calls, that with n equal to 3, tried the bogus plugin, rejected it, and then returned the next one.)
2. Clean module, successfully
3. Call vampGetPluginDescriptor(2, 0) again. This time isPythonInitialised 1 and haveScannedPlugins is back to 0. The scan happens with the same results as last time.
4. Call vampGetPluginDescriptor(2, 1). Both isPythonInitialised and haveScannedPlugins are 1. This plugin succeeds.
5. Call vampGetPluginDescriptor(2, 2). Both isPythonInitialised and haveScannedPlugins are 1. This plugin succeeds.
6. Call vampGetPluginDescriptor(2, 3). This fails to load the bogus plugin, then returns the next plugin in the list.
7. Clean module, successfully
8. Call vampGetPluginDescriptor(2, 0) for a third time. Again isPythonInitialised 1 and haveScannedPlugins is back to 0. The scan happens with the same results as previously.
9. Call vampGetPluginDescriptor(2, 1). Both isPythonInitialised and haveScannedPlugins are 1. This plugin succeeds.
10. Call vampGetPluginDescriptor(2, 2). Both isPythonInitialised and haveScannedPlugins are 1. This plugin succeeds.
11. Clean the module, which crashes.

#4 - 2019-01-11 03:29 PM - Chris Cannam

Note: I was using the Piper server for testing because I wanted to follow what Sonic Visualiser is doing, but it's also possible to reproduce this by just running vamp-simple-host -l. There does seem to be some order dependency - whether the crash happened or not for me depended on precisely which other VamPy plugins were in the directory alongside my test file. With PyMFS and my test file, vamp-simple-host -l always crashes. The crash happens just after PyMFS has been listed to the output, meaning this is a load/query/unload cycle in which the test file was not even requested.

#5 - 2019-01-14 03:51 PM - Chris Cannam

First thought from GF:

"Vampy itself exports the class symbols: Feature, FeatureList and FeatureSet, and therefore these should be reserved names in a Vampy plugin.

"The crash is probably caused when cleanModule is deleting references to instances created in the python script (and therefore already deleted when the execution is finished) rather than instances of the structures exported by vampy.

"A good solution would be to check if a VamPy plugin redefines any of the reserved symbols and avoid loading the plugin if they do."

#6 - 2019-01-14 03:58 PM - Chris Cannam

Unfortunately if you just check for the existence of any of these names when scanning the plugin modules, and reject any module that has any of them defined, then you'll end up rejecting all Vampy plugins because they all have them defined by virtue of importing the vampy namespace.

Also, even if you reject the plugin at this point in the scan, its code has already been loaded, and so any names it uses will need to be cleared when (or before?) Vampy is unloaded (even if the plugin module is blacklisted and never used), or else you get a crash the **next** time the modules are scanned, because a stale name still lives in the previously-loaded module table.

#7 - 2019-01-14 04:22 PM - Chris Cannam

The approach I've taken in commit:a6718f9fe942 is to check on scan whether a name has been redefined by the module, and refuse to load it if so; and also, reset any refused module's class dictionary to None at the point where loading is refused, so that we don't need to clean it on unload (since we have no record of it then).

This seems convincing and appears to work, but I need to give it a few more tests.